

The University of Texas at Tyler  
Bachelor of Science in Computer Science

## Syllabus

<b>Course Number:</b>	COSC 2336
<b>Course Title:</b>	Data Structures and Algorithms
<b>Course Description:</b>	Topics include recursion, the underlying philosophy of object-oriented programming, fundamental data structures (including stacks, queues, linked lists, hash tables, trees, and graphs), the basics of algorithmic analysis, and an introduction to the principles of language translation.
<b>Pre-requisites:</b>	MATH 2330, COSC 1337/1137
<b>Credits:</b>	3
<b>Text(s):</b>	Frank M. Carrano, Janet Prichard, Data Abstraction and Problem Solving with Java, Walls and Mirrors, Second Edition, Addison-Wesley, 2006.
<b>Languages Used: (if applicable)</b>	Java (students are also allowed to submit assignments in C/C++)
<b>Topics:</b>	<ol style="list-style-type: none"> <li>1. Review of elementary programming concepts</li> <li>2. Stacks, queues, linked lists, hash tables, trees and graphs</li> <li>3. Review of object-oriented programming</li> <li>4. Sorting and searching techniques</li> <li>5. Recursion</li> <li>6. Basic algorithmic analysis</li> <li>7. Algorithmic strategies: brute-force; greedy; divide-and-conquer; backtracking; branch-and-bound; heuristics; pattern matching; and numerical approximation algorithms</li> <li>8. Overview of programming languages and programming paradigms</li> <li>9. Software validation; testing and object-oriented testing</li> </ol>
<b>Additional Materials:</b>	

<b>Evaluation Method: (only items in dark print apply)</b>	
<b>1. Examination/Quiz</b>	<b>2. Homework</b>
3. Paper/Report	<b>4. Computer Program</b>
5. Project	6. Presentation
<b>7. Class Participation</b>	8. Peer Review

<b>Course Objectives<sup>1</sup>: By the end of this course students are expected to:</b>
1. Illustrate by example the basic terminology of graph theory, and some of the properties and special cases of each. [1,2,4,7]

2. Demonstrate different traversal methods for trees and graphs. [1,2,4,7]
3. Model problems in computer science using graphs and trees. [1,2,4,7]
4. Discuss the representation and use of primitive data types and built-in data structures. [1,2,4,7]
5. Describe how the data structures in the topic list are allocated and used in memory. [1,2,4,7]
6. Describe common applications for each data structure in the topic list. [1,2,4,7]
7. Implement the user-defined data structures in a high-level language. [1,2,4,7]
8. Compare alternative implementations of data structures with respect to performance. [1,2,4,7]
9. Write programs that use each of the following data structures: arrays, records, strings, linked lists, stacks, queues, and hash tables. [1,2,4,7]
10. Compare and contrast the costs and benefits of dynamic and static data structure implementations. [1,2,4,7]
11. Choose the appropriate data structure for modeling a given problem. [1,2,4,7]
12. Describe the concept of recursion and give examples of its use. [1,2,4,7]
13. Identify the base case and the general case of a recursively defined problem. [1,2,4,7]
14. Compare iterative and recursive solutions for elementary problems such as factorial. [1,2,4,7]
15. Describe the divide-and-conquer approach. [1,2,4,7]
16. Implement, test, and debug simple recursive functions and procedures. [1,2,4,7]
17. Describe how recursion can be implemented using a stack. [1,2,4,7]
18. Discuss problems for which backtracking is an appropriate solution. [1,2,4,7]
19. Determine when a recursive solution is appropriate for a problem. [1,2,4,7]
20. Explain the use of big O, omega, and theta notation to describe the amount of work done by an algorithm. [1,2,4,7]
21. Use big O, omega, and theta notation to give asymptotic upper, lower, and tight bounds on time and space complexity of algorithms. [1,2,4,7]
22. Determine the time and space complexity of simple algorithms. [1,2,4,7]
23. Implement a greedy algorithm to solve an appropriate problem. [1,2,4,7]
24. Implement a divide-and-conquer algorithm to solve an appropriate problem. [1,2,4,7]
25. Use backtracking to solve a problem such as navigating a maze. [1,2,4,7]
26. Implement the most common quadratic and $O(N \log N)$ sorting algorithms. [1,2,4,7]
27. Design and implement an appropriate hashing function for an application. [1,2,4,7]
28. Design and implement a collision-resolution algorithm for a hash table. [1,2,4,7]
29. Discuss the computational efficiency of the principal algorithms for sorting, searching, and hashing. [1,2,4,7]
30. Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application specific patterns in the input data. [1,2,4,7]
<sup>1</sup> Numbers in bracket refer to method(s) used to evaluate the course objective.

<p><b>Relationship to Program Outcomes: (only items in dark print apply )<sup>2</sup></b>  <b>This course supports the following Computer Science Program Outcomes, which state that our students at the time of graduation are expected to:</b></p>	
<p><b>1. Posses knowledge of the fundamentals of mathematics, science, and technology.</b>  <b>[1,2,3,20,21,22,29]</b></p>	
<p><b>2. Be able to use modern computational tools and techniques in the practice of computer science.</b> [3,4,5,6,12,13,14,15,16,17,18,19,20,21,23,24,25,26,27,28,29,30].</p>	
<p><b>3. Be able to develop logically sound and efficient algorithms.</b>  <b>[3,4,8,9,10,11,12,13,14,15, 16,17,18,19,20,21,22,23,24,25,26,27,28,29]</b></p>	
<p><b>4. Be prepared to implement algorithms in multiple programming languages, on multiple hardware platforms, and in multiple operating system environments.</b>  <b>[7,16,26,27]</b></p>	
<p>5. Be able to perform analysis, design, implementation, testing, and maintenance of computer-based systems, stressing software engineering principles.</p>	
<p>6. Be prepared to seek continuing professional development, graduate studies, or professional certifications related to computer science.</p>	
<p>7. Demonstrate effective written, visual and oral communication skills.</p>	
<p>8. Posses an educational background to understand the global context in which computer science is practiced, including:</p> <ul style="list-style-type: none"> <li>a. Knowledge of contemporary issues related to computer science;</li> <li>b. The impact of computers on society;</li> <li>c. The role of ethics in the practice of computer science.</li> </ul>	
<p>9. Be able to contribute effectively as members of a project development team.</p>	
<p>10. Recognize the need to pursue continued learning throughout their professional careers.</p>	
<p><sup>2</sup>Numbers in brackets refer to course objective(s) that address the Program Outcome.</p>	

Prepared by: Stephen B. Rainwater	Date: 10/25/04
Modified by: Artur Mikitiuk	Revised: 01/13/05, 08/23/07

### **The Object-Oriented Paradigm (COSC 1137)**

<b>Course Objectives<sup>1</sup>: By the end of this course students are expected to:</b>
1. Justify the philosophy of object-oriented design and the concepts of encapsulation, abstraction, inheritance, and polymorphism. [1,2,4,7]
2. Design, implement, test, and debug simple programs in an object-oriented programming language. [1,2,4,7]
3. Describe how the class mechanism supports encapsulation and information hiding. [1,2,4,7]
4. Design, implement, and test the implementation of “is-a” relationships among objects using a class hierarchy and inheritance. [1,2,4,7]
5. Compare and contrast the notions of overloading and overriding methods in an object-oriented language. [1,2,4,7]
6. Explain the relationship between the static structure of the class and the dynamic structure of the instances of the class. [1,2,4,7]
7. Describe how iterators access the elements of a container. [1,2,4,7]

### **Discrete Structures (MATH 2330)**

<b>Course Objectives<sup>1</sup>: By the end of this course students are expected to:</b>
1. Relate graphs and trees to data structures, algorithms, and counting. [1,2,4,7]
2. Solve elementary recurrence relations. [1,2,4,7]

### **Other – Programming Languages, Compiler Techniques, Software Development, Analysis of Algorithms**

<b>Course Objectives<sup>1</sup>: By the end of this course students are expected to:</b>
1. Use numerical approximation to solve mathematical problems, such as finding the roots of a polynomial. [1,2,4,7]
2. Discuss the concept of finite state machines. [1,2,4,7]
3. Explain context-free grammars. [1,2,4,7]
4. Design a deterministic finite-state machine to accept a specified language. [1,2,4,7]
5. Explain how some problems have no algorithmic solution. [1,2,4,7]
6. Provide examples that illustrate the concept of uncomputability. [1,2,4,7]
7. Summarize the evolution of programming languages illustrating how this history has led to the paradigms available today. [1,2,4,7]
8. Identify at least one distinguishing characteristic for each of the programming paradigms covered in this unit. [1,2,4,7]
9. Evaluate the tradeoffs between the different paradigms, considering such issues as space efficiency, time efficiency (of both the computer and the programmer), safety, and power of expression. [1,2,4,7]
10. Distinguish between programming-in-the-small and programming-in-the-large. [1,2,4,7]
11. Distinguish between program validation and verification. [1,2,4,7]
12. Describe the role that tools can play in the validation of software. [1,2,4,7]

13. Distinguish between the different types and levels of testing (unit, integration, systems, and acceptance) for medium-size software products. [1,2,4,7]
14. Create, evaluate, and implement a test plan for a medium-size code segment. [1,2,4,7]
15. Undertake, as part of a team activity, an inspection of a medium-size code segment. [1,2,4,7]
16. Discuss the issues involving the testing of object-oriented software. [1,2,4,7]
17. Deduce recurrence relations that describe the time complexity of recursively defined algorithms. [1,2,4,7]
18. Describe the shortcoming of brute-force algorithms. [1,2,4,7]
19. For each of several kinds of algorithm (brute force, greedy, divide-and-conquer, backtracking, branch-and-bound, and heuristic), identify an example of everyday human behavior that exemplifies the basic concept. [1,2,4,7]
20. Describe various heuristic problem-solving methods. [1,2,4,7]
21. Use pattern matching to analyze substrings. [1,2,4,7]
22. Solve problems using the fundamental graph algorithms, including depth-first and breadth-first search, single-source and all-pairs shortest paths, transitive closure, topological sort, and at least one minimum spanning tree algorithm. [1,2,4,7]
23. Demonstrate the following capabilities: to evaluate algorithms, to select from a range of possible options, to provide justification for that selection, and to implement the algorithm in programming contexts. [1,2,4,7]